ELSEVIER

Computational Geometry 20 (2001) 39–52

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Spirale Reversi: Reverse decoding of the Edgebreaker encoding

Martin Isenburg, Jack Snoeyink *

Department of Computer Science, College of Arts and Sciences, University of North Carolina at Chapel Hill, Campus Box 3175, Sitterson Hall, Chapel Hill, NC 27599-3175, USA

Communicated by D. Bremner; received 29 August 2000; accepted 30 November 2000

Abstract

We present a simple linear time algorithm for decoding Edgebreaker encoded triangle meshes in a single traversal. The Edgebreaker encoding technique, introduced by Rossignac (1999), encodes the connectivity of triangle meshes homeomorphic to a sphere with a guaranteed 2 bits per triangle or less. The encoding algorithm visits every triangle of the mesh in a depth-first order. The original decoding algorithm recreates the triangles in the same order they have been visited by the encoding algorithm and exhibits a worst case time complexity of $O(n^2)$. More recent work (Rossignac and Szymczak, 1999) uses the same traversal order and improves the worst case to $O(n)$. However, for meshes with handles multiple traversals are needed during both encoding and decoding. We introduce here a simpler decoding technique that performs a single traversal and recreates the triangles in reverse order.

Keywords: Connectivity compression; Edgebreaker; Linear decoding

1. Introduction

Efficiently encoding the connectivity of triangular meshes has recently been subject of intense study [2–5,7,10,11] and many representations have been proposed. The sudden interest in this area is fueled by the emerging demand for transmitting 3D data sets over the Internet. Since transmission bandwidth is a scarce resource, compact encodings for 3D models are of great advantage.

The Edgebreaker encoding technique, introduced in [7], encodes the connectivity of triangle meshes homeomorphic to a sphere with a guaranteed 2 bits per triangle or less. The encoding algorithm visits each triangle of the mesh in a depth-first order using five different operations called C, L, E, R and S. Each triangle is labeled according to the operation that processes it. The resulting *CLERS string* is a compact encoding of the connectivity of the mesh.

The original decoding algorithm [7] recreates the triangles in the same order as they have been visited by the encoding algorithm. This decoding algorithm has an asymptotic worst case time

* Corresponding author.
E-mail addresses: isenburg@cs.unc.edu (M. Isenburg), snoeyink@cs.unc.edu (J. Snoeyink).

0925-7721/01/$ – see front matter
PII: S0925-7721(01)00034-7

complexity of $O(n^2)$. These costs are a result of the look-ahead procedure that is necessary for decoding subsequences in the CLERS sequence. These subsequences, which are encapsulated by an S operation and a corresponding E operation, reflect recursions in the Edgebreaker encoding scheme. More recent work called Wrap&Zip [8] eliminates the need for this look-ahead procedure and improves the worst case time complexity to $O(n)$. However, this algorithm requires multiple traversals of the mesh triangles for meshes with handles and an initial traversal of the CLERS string for meshes with boundary.

We introduce here a simpler decoding technique which recreates the triangles in reverse order. The CLERS sequence is processed backwards starting at the last label. This completely eliminates the look-ahead procedure of [7] or the zipping procedure of [8]. Following a suggestion by Jarek Rossignac we call this decoding scheme Spirale Reversi.

In the next section we explain the Edgebreaker encoding scheme. A detailed description of the algorithm can be found in [7]. The original Edgebreaker decoding scheme is covered in Section 3 and the Wrap&Zip decoding scheme in Section 4. We introduce our Spirale Reversi decoding scheme in Section 5. These sections describe encoding and decoding only for simple meshes. We explain how the algorithms generalize to meshes with boundary in Section 6, with holes in Section 7 and with handles in Section 8.

2. Edgebreaker encoding

Before we describe the Edgebreaker encoding scheme, we define what properties the input mesh is expected to have:

(1) The mesh is a surface composed of topological triangles (i.e., every face is bound by three edges).
(2) The mesh has no boundary and no holes (i.e., every edge is bound by two faces).
(3) The mesh has no handles (i.e., the mesh is topologically equivalent to a sphere).
(4) The mesh is 2-manifold (i.e., the local surface around every vertex is homeomorphic to a disk).

The Edgebreaker encoding process starts with a triangulated mesh and produces a CLERS string. It visits every triangle of the mesh by including it into an *active boundary.* Initially the active boundary is an arbitrary triangle of the mesh. The encoding uses five different operations called C, L, E, R and S to include a triangle into the active boundary. Which operation is chosen depends on how the respective triangle is attached to the active boundary at the moment it is processed. This expands (operation C), shrinks (operation R and L), splits (operation S) or ends (operation E) the active boundary. The CLERS string that describes the sequence of traversal operations is a compact encoding of the mesh connectivity. Now the details:

The encoding process starts off with picking an arbitrary triangle of the mesh as the initial active boundary. It has three *boundary edges,* which are directed clockwise around the triangle. The triangle itself is declared to be *inside,* the remaining mesh to be *outside* of the boundary. One of the three initial boundary edges is defined to be the *gate* of the boundary. The gate is directed in the same way as the boundary edges. The triangle right of the gate is inside, the triangle left of the gate is outside of the boundary. The *active gate* is the gate of the active boundary. The *active triangle* is the triangle left of the active gate. An initially empty stack is used to temporarily store boundaries.

With every operation of the encoding process the active triangle moves from outside to inside of the active boundary. A triangle that lies outside of some boundary is not yet encoded. A triangle that lies inside of all boundaries is already encoded. This process terminates after $t - 1$ operations, with t being

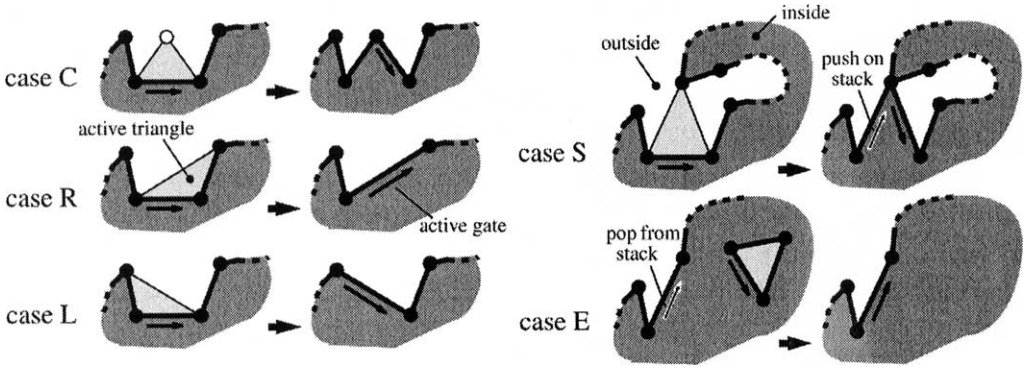


Fig. 1. The Edgebreaker encoding operations C, R, L, S and E.

the number of mesh triangles. Each triangle is included into the active boundary by one operation – except the triangle that defines the initial active boundary.

Which operation is chosen to include the active triangle depends on how it is attached to the active boundary (see Fig. 1). If its third vertex is not on the active boundary then operation C is used. If its third vertex is the next boundary vertex on the active boundary then operation R is used. (Remember that the boundary edges are directed clockwise around the inside.) If its third vertex is the previous boundary vertex on the active boundary then operation L is used. If its third vertex is some other boundary vertex on the active boundary then operation S is used. If its third vertex is the previous *and* the next boundary vertex on the active boundary then operation E is used. This can only happen for an active boundary of length three.

With each operation the active boundary and the active gate are updated. These updates are as follows:

- The *C operation* inserts two and removes one boundary edge. The old gate is the removed boundary edge, the new gate is the inserted boundary edge right of the old gate.
- The *R and L operation* both insert one and remove two boundary edges. The new gate is the inserted boundary edge, the old gate is one of the removed boundary edges. The two operations differ by whether the old gate is on the right (R) or on the left (L) as seen from the new gate.
- The *S operation* splits the active boundary into two boundaries that share a vertex. It inserts two and removes one boundary edge. Both inserted boundary edges become gates for the boundary they belong to. The one left of the old gate is pushed on the stack. The other becomes the active boundary.
- The *E operation* removes the last three boundary edges. If the stack is empty the encoding process terminates, otherwise it continues on a boundary popped from the stack.

The example in Fig. 8 leads step by step through the final twelve operations of Edgebreaker encoding a mesh.

For triangle meshes with v vertices and t triangles that are homeomorphic to a sphere t equals $2v - 4$. The traversal of the mesh triangles reaches new vertices only with the C operation. Since there are two times more triangles than vertices, half of all operations will be of type C. A straight-forward compression scheme that codes a C operation with one bit and the remaining four operations with three bits is guaranteed to use no more than $2t$ or $4v$ bits. More elaborate compression schemes for the CLERS sequence guarantee even lower bounds of $3.67v$ bits [6] or $3.55v$ bits [1].

3. Edgebreaker decoding

The Edgebreaker decoding process starts with a CLERS string and produces a triangulated mesh. Two traversals of the CLERS string are needed: A preprocessing phase that computes offset values. And a generation phase that creates the triangles in the order in which they were encoded by the Edgebreaker encoding process.

The preprocessing phase computes an offset value for every S operation. The Edgebreaker encoding uses the S operation whenever the third vertex of the active triangle is a vertex on the active boundary other than the previous or the next. When the Edgebreaker decoding creates this triangle, it needs to know which vertex on the active boundary to use as the triangle's third vertex. The offset value that is computed in this preprocessing phase is the distance (i.e., the number of vertices) between the active gate and this vertex along the active boundary.

The computation of these offset values is simple. The resulting change in boundary length is added up for all operations following an S operation until and including its corresponding E operation. Since pairs of S and E operations are always nested, the offset values for all S operations can be computed in a single traversal (see Fig. 2).

The generation phase starts with creating the initial triangle. The active boundary and the gate are identified and the CLERS string is processed. What follows is an almost exact replay of the encoding algorithm. With every operation a new triangle is created and included into the active boundary. The triangle is always attached to the left of the active gate. Which vertex is used as the triangle's third vertex depends on the current operation. Only for the C operation a new vertex is created. For all other operations a vertex from the active boundary is used. For the R operation this is the next and for the L operation the previous vertex on the active boundary. For the S operation it is some other boundary vertex. The precomputed offset value specifies its distance from the active gate along the boundary. When the E operation occurs, the active boundary consists of only three boundary edges – leaving no choice for the third vertex.

The operations C, R, L and E of Edgebreaker decoding perform the same updates on boundary and gate as during encoding (see Fig. 1), only operation S is more complex since it needs to use the precomputed

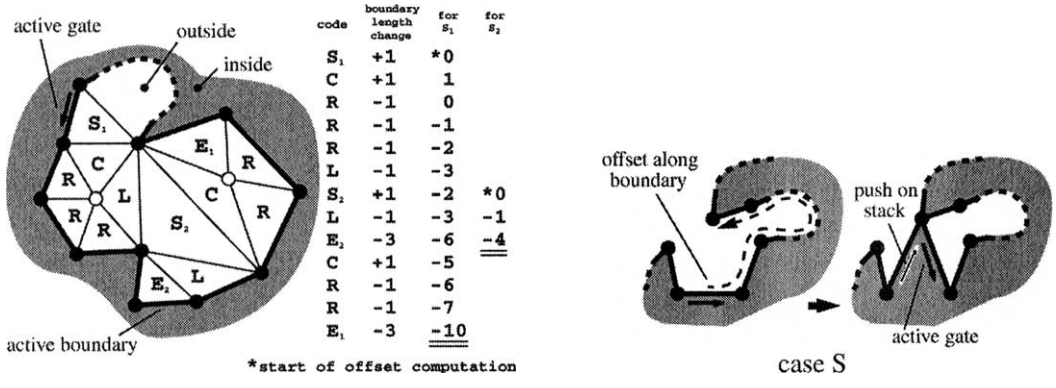


Fig. 2. Computing an offset for each S operation during the preprocessing phase (a) and using such an offset during the generation phase (b).

offset (see Fig. 2). The example in Fig. 9 leads step by step through the final twelve operations of Edgebreaker decoding a mesh.

Although in practice only a small fraction of operations are of type S, they imply an asymptotic worst case time complexity of $O(n^2)$ for the Edgebreaker decoding, if the active boundary is maintained in a linear data structure (such as a double linked list). Each S operation requires a linear search for the vertex specified by the offset. This cost may be reduced to $O(n \log n)$ if the active boundary is maintained in a data structure with a logarithmic instead of a linear search time. However, the more complex update operations of a data structure with logarithmic search time (such as a balanced binary tree) would increase the expected time complexity from $O(n)$ to $O(n \log n)$.

4. Wrap&Zip decoding

The Wrap&Zip decoding process starts with a CLERS string and produces a triangulated mesh. Only one traversal of the CLERS string is needed. It starts with creating three vertices that form the initial triangle. The active boundary and the gate are identified and the CLERS string is processed. What follows is a modified replay of the encoding algorithm. With every operation a new triangle is created. This triangle is always attached to the left of the active gate. But the decision which vertex is the triangle's third vertex is postponed for the operations R, L, S and E. Only for the C operation a newly created vertex is used. This is the wrapping part – during the zipping part the unlabeled vertices will eventually be identified with a previously created vertex.

All boundary edges except for the gate have an additional *zip direction* assigned that depends on the operation that created them. Which operation assigns which zip direction to which edge is shown in Fig. 3. Each time the zip directions of two adjacent boundary edges point to a common vertex, they are zipped together by identifying their other ends. This zipping continues recursively if the resulting vertex exhibits the same property. Whether a zip is necessary needs only to be checked after L and E operations. No immediate zipping is possible after C, R and S operations. A zip after an L operation never starts recursive zipping, whereas a zip after an E operation always does. In Fig. 4 we illustrate single zipping after an L operation and recursive zipping after an E operation.

The example in Fig. 10 leads step by step through the final twelve operations of Wrap&Zip decoding a mesh.

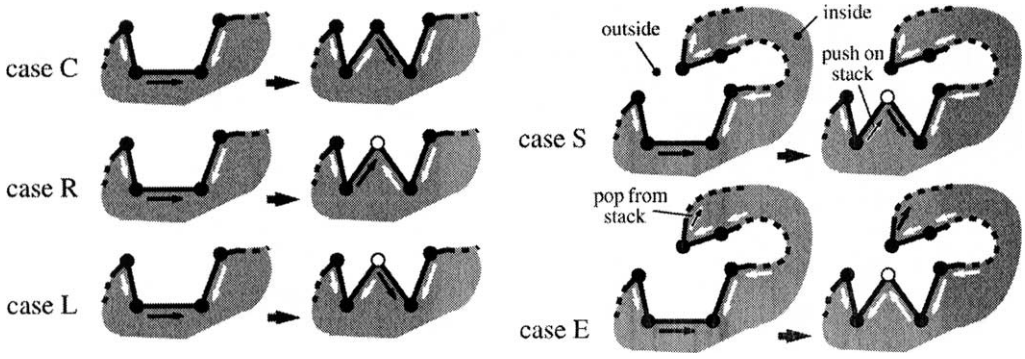
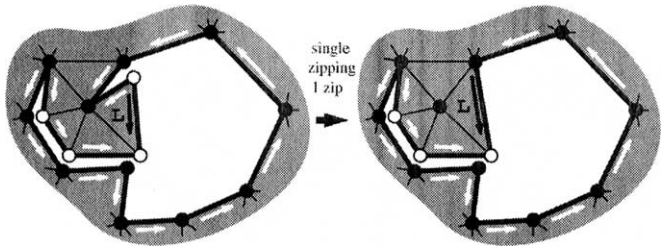
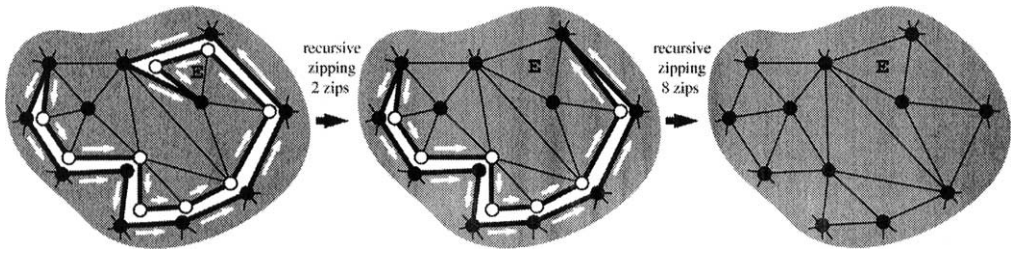


Fig. 3. The Wrap&Zip decoding operations. White arrows denote assigned zip directions.



(a)



(b)

Fig. 4. Single zipping after an L operation (a) and recursive zipping after an E operation (b).

The wrapping and zipping technique improves on the worst case time complexity of the Edgebreaker decoding as it avoids the vertex search for the S operation. It can be shown that the number of zip operations equals the number of edges in the vertex-spanning tree. Therefore the Wrap&Zip decoding algorithm has linear time complexity.

5. Spirale Reversi decoding

The Spirale Reversi decoding process starts with a CLERS string and produces a triangulated mesh. Only one *reverse* traversal of the CLERS string is needed. This completely eliminates the look-ahead procedure of [7] or the zipping procedure of [8] involved with the S and E operation pairs. It can be seen as a step by step reversal of the Edgebreaker encoding process.

The Spirale Reversi decoding scheme uses similar boundary definitions as the Edgebreaker encoding scheme. It starts with creating a triangle with three unlabeled vertices as the initial boundary. The boundary edges are directed counterclockwise around this triangle, which is declared to be outside of the boundary. One of the three boundary edges is picked as the initial active gate. Inside of the boundary is right of the gate, outside of the boundary is left of the gate. The Edgebreaker encoding was growing the inside until there was no unencoded triangle left outside. The Spirale Reversi decoding is growing the outside until there is no undecoded triangle left inside. This reflects the *reverseness* of the Spirale Reversi decoding.

With every operation of the Spirale Reversi decoding scheme the triangle left of the active gate moves from inside to outside of the active boundary. A triangle that lies outside of some boundary is already decoded. A triangle that lies inside of all boundaries is not yet decoded. The CLERS sequence

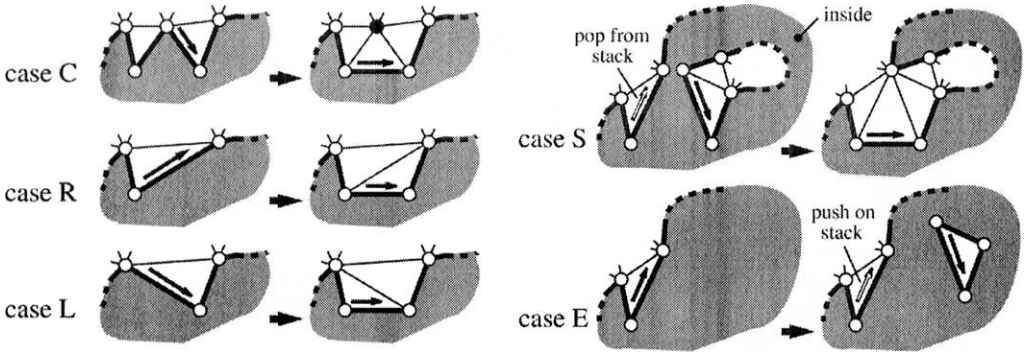


Fig. 5. The Spirale Reversi decoding operations.

is processed in reverse order. Depending on the operation the active boundary is shrunk (operation C), is expanded (operation R and L), is merged with a stack boundary (operation S), or is created new (operation E).

Reversing the encoding algorithm works as follows: Each operation creates a new triangle. For operations C, R, L and S this triangle is attached to the right of the active gate. Which vertex is used as the triangle's third vertex depends on the operation (see Fig. 5). For the C operation it is the previous vertex along the active boundary. The R and the L operation use a new unlabeled vertex. For the S operation a vertex from a boundary popped from the stack is used. More exactly it is the vertex at the origin of this boundary's gate. Simultaneously the vertex at the destination of this gate and the vertex at the origin of the active gate are identified. For operation E a new triangle with three unlabeled vertices is created that is not attached to anything previously decoded. The updates of the boundary and the gate are as follows:

- The *C operation* removes two and inserts one boundary edge. The new gate is the inserted boundary edge, the old gate is the removed boundary edge right of the new gate.
- *The R and L operation* both remove one and insert two boundary edges. The old gate is the removed boundary edge, the new gate is one of the inserted boundary edges. The two operations differ by whether the new gate is on the right (R) or on the left (L) as seen from the old gate.
- The *S operation* merges the active boundary with a boundary that is popped from the stack, thereby identifying one of their vertices. This removes two and inserts one boundary edge. Both removed boundary edges are old gates of the respective boundary. The new gate is the inserted boundary edge.
- The *E operation* creates a new active boundary with three boundary edges, one of which is the new gate. The current active boundary is pushed on the stack.

The example in Fig. 11 leads step by step through the first twelve operations of Spirale Reversi decoding a mesh.

We use a half-edge structure to store the mesh connectivity and to maintain the boundaries during decoding. Besides pointers to the origin, the next half-edge, and the inverse half-edge, we have two pointers to reference a next and a previous boundary edge. This way we organize all half-edges of the same boundary into a cyclic double linked list. Since each operation performs only a constant number of pointer updates, Spirale Reversi decodes the triangle mesh connectivity in linear time.

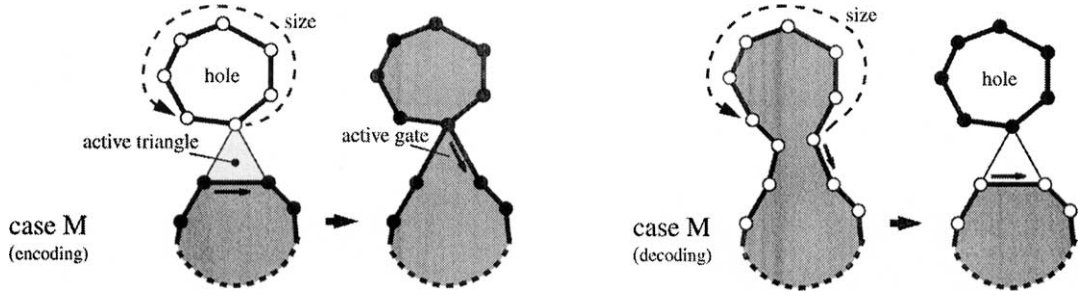


Fig. 6. Encoding a hole with Edgebreaker (a) and decoding of this hole with Spirale Reversi (b).

6. Encoding meshes with boundary

A triangle mesh with a single hole is often referred to as a mesh with boundary. A small variation makes Edgebreaker capable of encoding meshes with a single hole: Instead of the loop of edges around a mesh triangle we use the loop of edges around the hole as the initial active boundary. An arbitrary edge from this boundary is declared to be the initial active gate and encoding proceeds as before.

Both the Edgebreaker decoding and the Wrap&Zip decoding need additional information to decode a mesh with boundary. They need to know the length of the initial active boundary (i.e., the size of the hole). This can be precomputed during an initial traversal of the CLERS string. The Spirale Reversi decoding needs no additional information. After decoding the last label of the reversed CLERS string, the active boundary loops around the hole.

7. Encoding meshes with holes

For every additional hole the Edgebreaker encoding runs into a situation in which the third vertex of the active triangle lies on the boundary of a hole. For this scenario the M operation was introduced. The active boundary is merged with the boundary of the hole by opening and joining both loops at their common vertex as depicted in Fig. 6. The label M and the *size* of the hole (i.e., the number of vertices/edges around the hole) are recorded.

The decoding of a hole is straight-forward for all three decoding algorithms. The Edgebreaker decoding and the Wrap&Zip decoding replay what happens during encoding and merge the active boundary with the hole boundary. The *size* value associated with label M specifies the size of this hole. For the Spirale Reversi decoding this value specifies how much of the active boundary needs to be pinched off to recreate the hole.

8. Encoding meshes with handles

For every mesh handle the Edgebreaker encoding eventually runs into a situation in which the third vertex of the active triangle is not on the active boundary, but on some other boundary in the stack. For

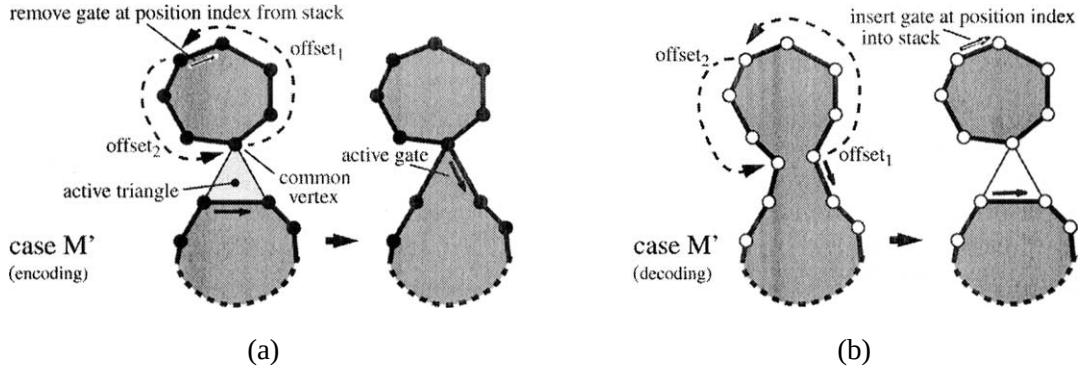


Fig. 7. Encoding a handle with Edgebreaker (a) and decoding this handle with Spirale Reversi (b).

this scenario the M′ operation was introduced. The two boundaries are merged by opening and joining them at their common vertex as shown in Fig. 7. The encountered boundary is removed from the stack and three integer values are recorded:

- The former stack position *index* of the removed boundary.
- The counterclockwise distance $offset_1$ from the common vertex to the gate of the encountered boundary.
- The distance $offset_2$ back to the common vertex.

We changed the original Edgebreaker encoding by the last integer – this allows us to decode a mesh with handles in a single traversal of the CLERS string.

The original Edgebreaker decoding uses the three integers associated with the M′ operation to replay the situation encountered during the encoding. The decoding cost per M′ operation is $O(n)$. Neither Wrap&Zip nor Spirale Reversi decoding aim at improving this time complexity. The number of M′ operations is bounded by the genus of the mesh and is generally small. Decoding meshes with handles using Wrap&Zip requires a modified Edgebreaker encoding that performs three instead of one traversal of the mesh triangles. For details we refer to the original reference [8].

The Spirale Reversi decoding of a handle follows the concept of reversing the encoding operation M′. The two offsets specify how much of the active boundary is pinched off and which boundary edge is used as the gate of the resulting boundary. The index specifies the position at which this boundary is inserted into the stack.

9. Discussion

We presented a simple linear time algorithm for decoding Edgebreaker encoded triangle meshes. The concept of reversing the encoding process allows the decoding of a mesh with a single traversal of the CLERS string. For simple meshes our scheme eliminates the need for the look-ahead procedure used by the original Edgebreaker decoding [7] and the zipping procedure used by the Wrap&Zip decoding [8]. For meshes with boundary and/or handles our scheme eliminates the need for multiple traversals of the CLERS string and/or the mesh triangles.

Previously suggested compression schemes for mapping the CLERS string into a compact bit-stream store the labels in forward order [1,6,8]. Then Spirale Reversi would need to reverse the CLERS string

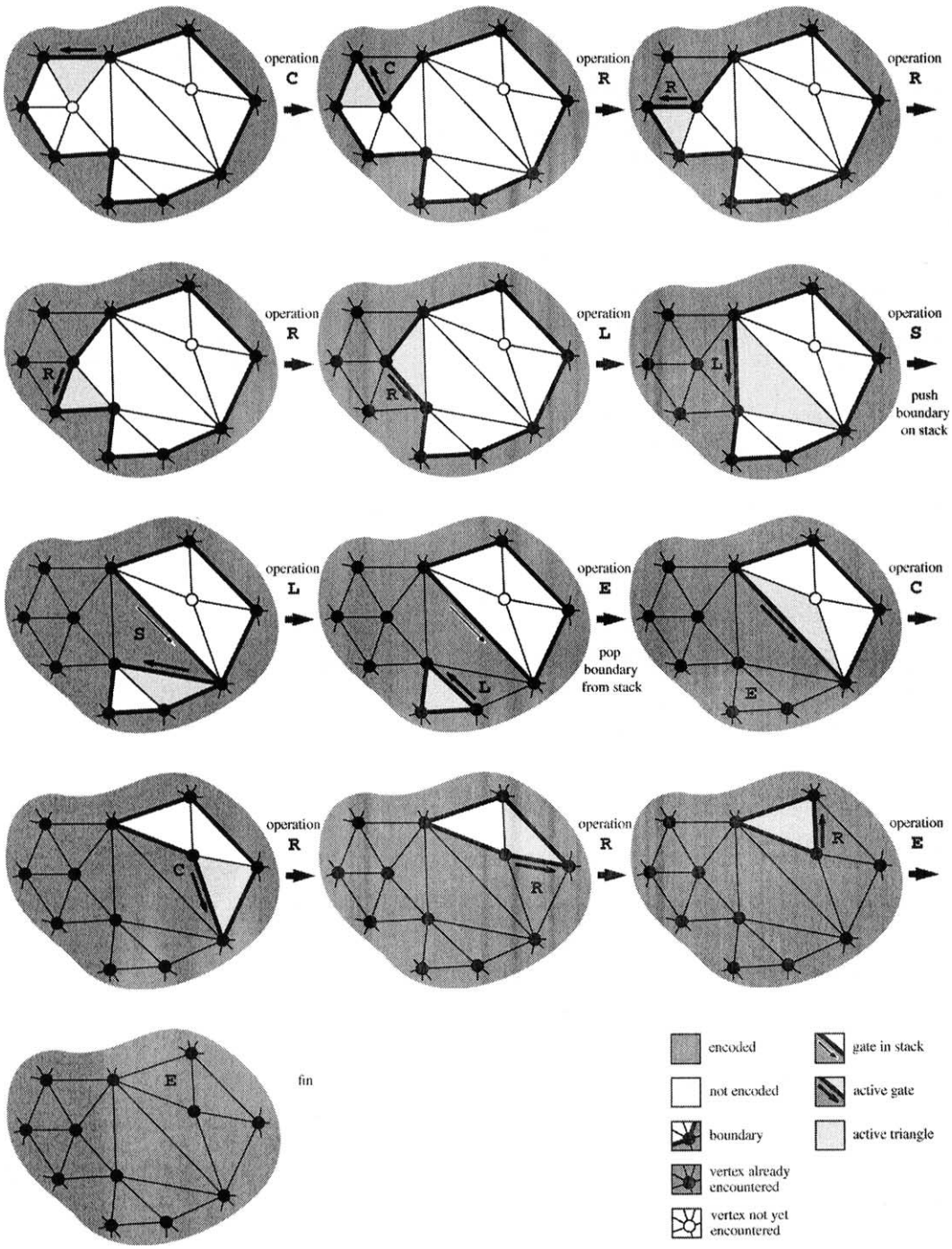


Fig. 8. An example of the final twelve operations of *Edgebreaker encoding*.

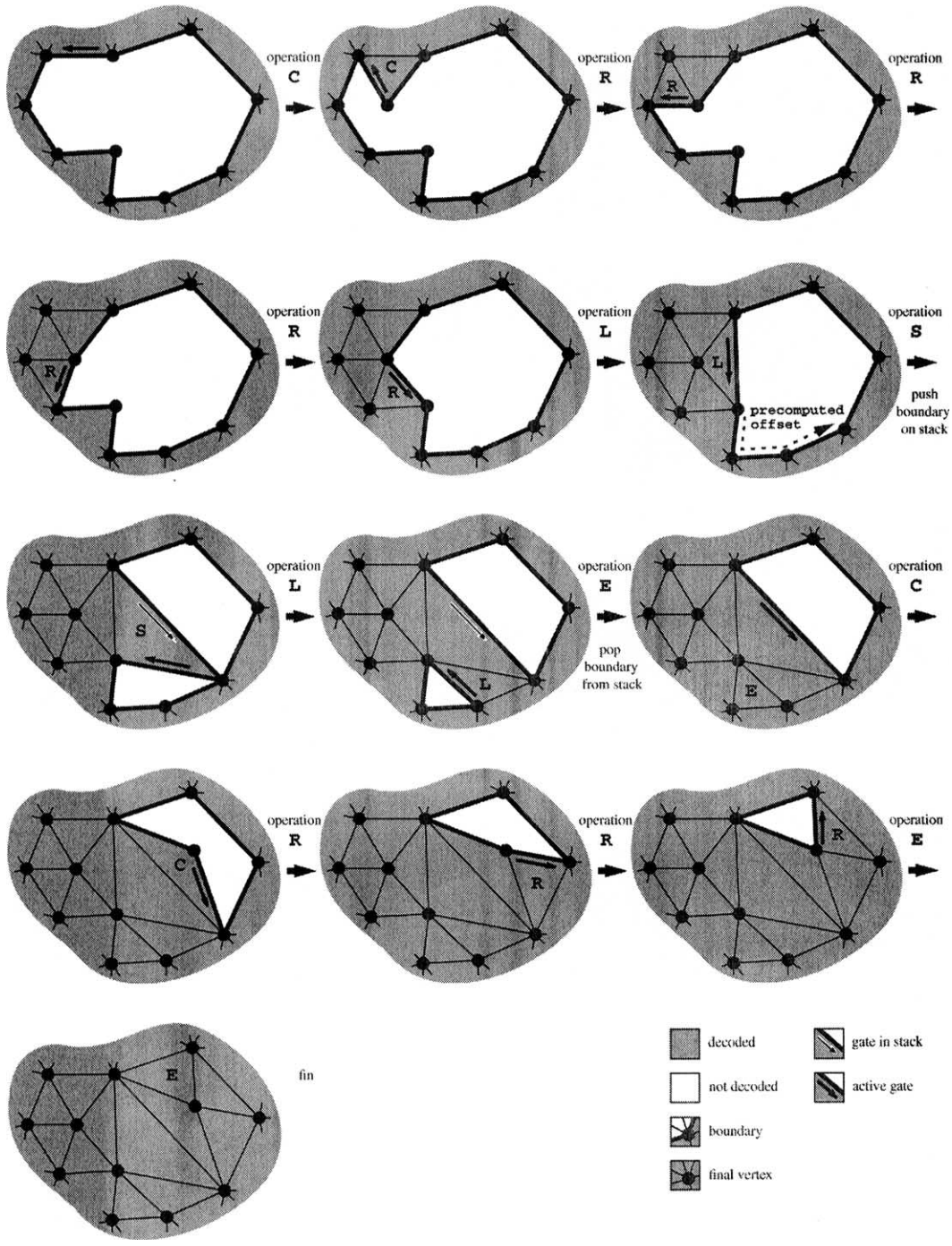


Fig. 9. An example of the final twelve operations of *Edgebreaker decoding*.

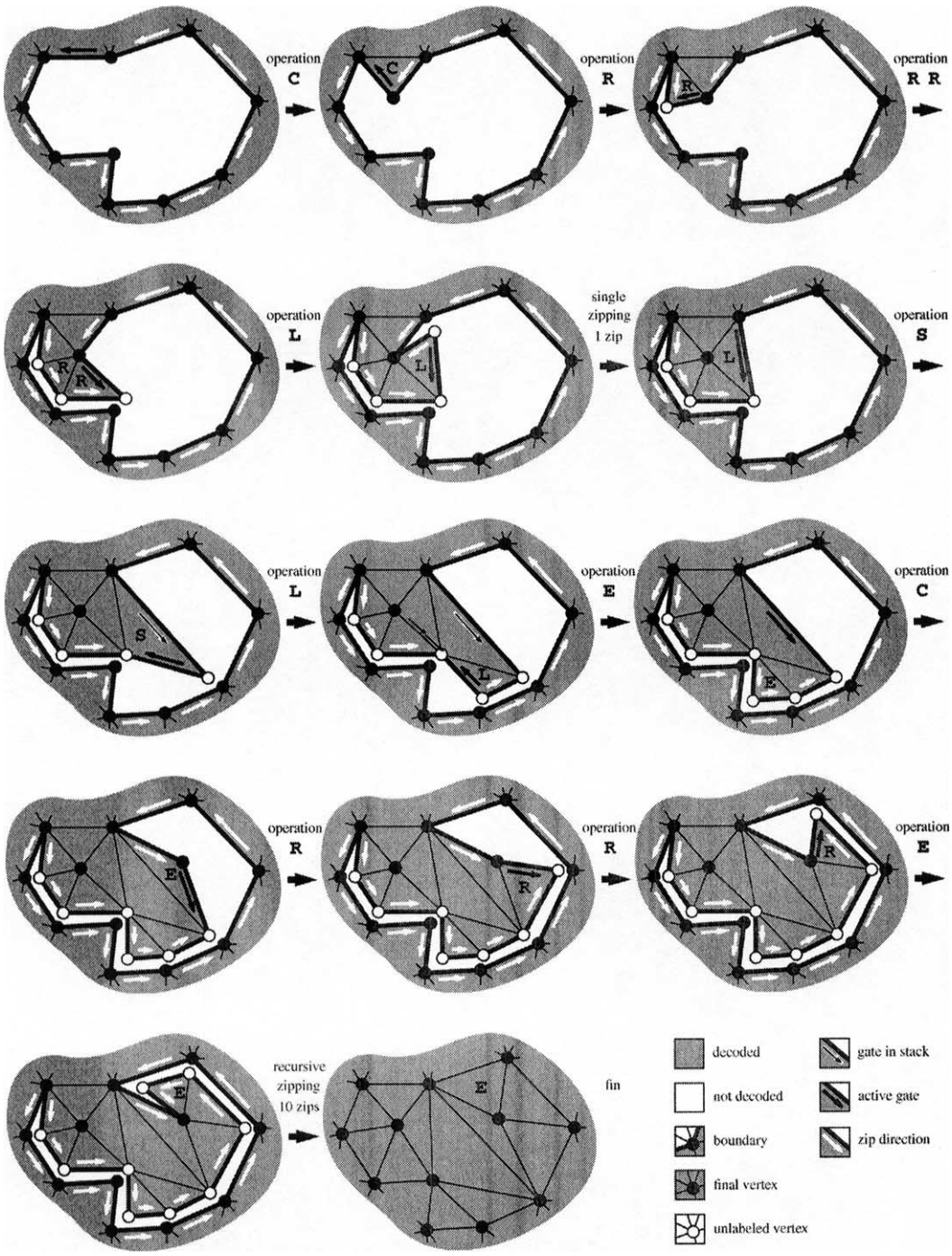


Fig. 10. An example of the final twelve operations of *Wrap&Zip decoding*.

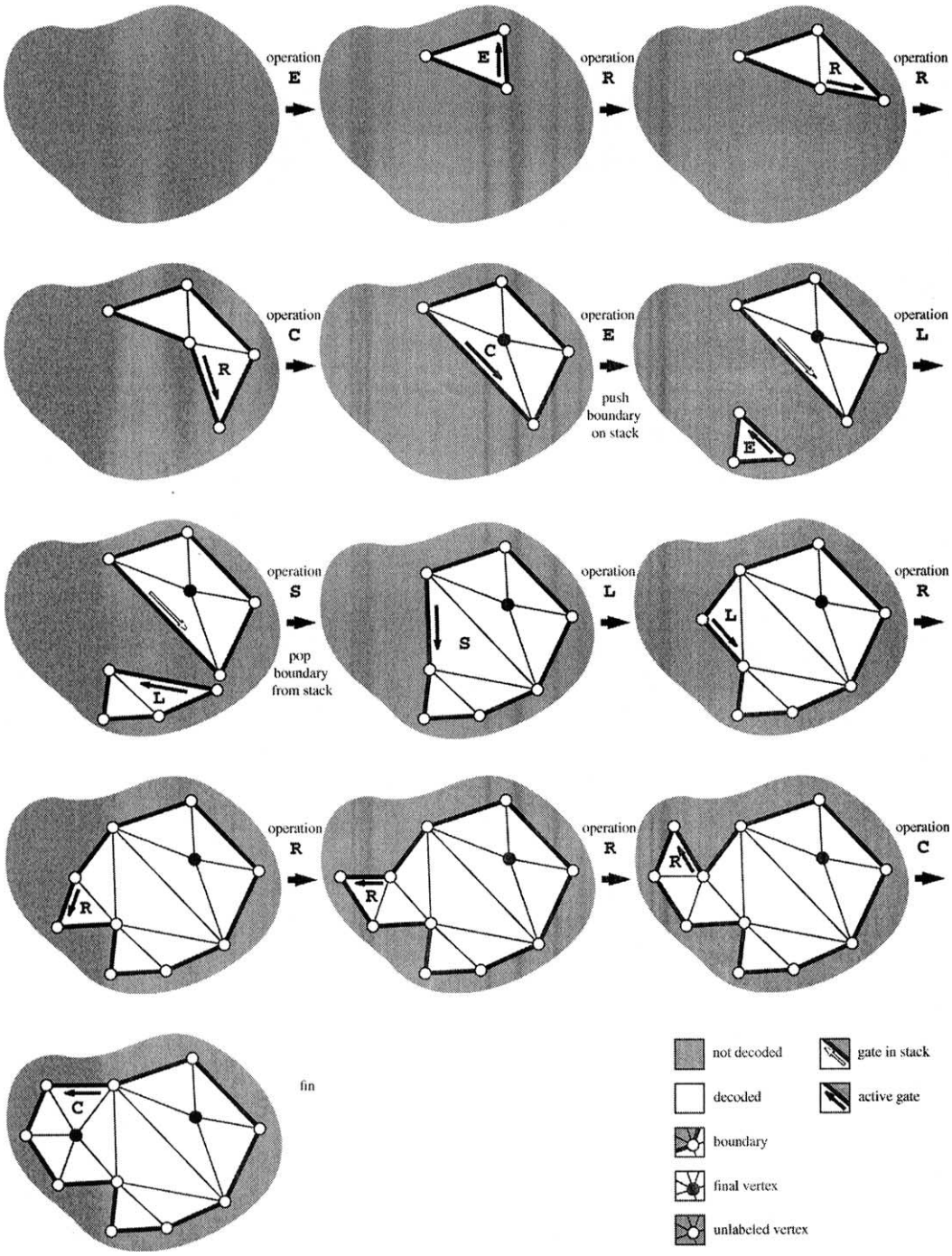


Fig. 11. An example of the first twelve operations of *Spirale Reversi decoding.*

first. However, the order-n entropy of the labels is about the same in both directions. This means that, for example, an arithmetic coder compresses the reversed CLERS string just as compactly as the unreversed one. Furthermore, for triangle meshes containing mainly vertices of degree six recent work by Szymczak et al. [9] exploits the reverseness of Spirale Reversi for efficient predictive compression of the labels.

Acknowledgements

The first author thanks Davis King for explaining the details of Edgebreaker during the *pool session* at SCG'99, Miami Beach, Florida, and Jarek Rossignac for suggesting the name Spirale Reversi. Most of this work was done while the authors were at the University of British Columbia (UBC) in Vancouver. It was supported by NSERC, IRIS, and a UBC Graduate Fellowship.

References

[1] S. Gumhold, New bounds on the encoding of planar triangulations. Technical Report WSI-2000-1, Wilhelm-Schikard-Institut für Informatik, Tübingen, March 2000.
[2] S. Gumhold, W. Strasser, Real time compression of triangle mesh connectivity, in: SIGGRAPH'98 Conference Proceedings, 1998, pp. 133–140.
[3] M. Isenburg, Triangle strip compression, in: Graphics Interface'2000 Conference Proceedings, 2000, pp. 197–204.
[4] M. Isenburg, J. Snoeyink, Mesh collapse compression, in: Proceedings of SIBGRAPI'99 – 12th Brazilian Symposium on Computer Graphics and Image Processing, 1999, pp. 27–28.
[5] M. Isenburg, J. Snoeyink, Face Fixer: Compressing polygon meshes with properties, in: SIGGRAPH'2000 Conference Proceedings, 2000, pp. 263–270.
[6] D. King, J. Rossignac, Guaranteed 3.67v bit encoding of planar triangle graphs, in: Proceedings of 11th Canadian Conference on Computational Geometry, 1999, pp. 146–149.
[7] J. Rossignac, Edgebreaker: Connectivity compression for triangle meshes, IEEE Transactions on Visualization and Computer Graphics 5 (1) (1999) 47–61.
[8] J. Rossignac, A. Szymczak, Wrap&Zip decompression of the connectivity of triangle meshes compressed with Edgebreaker, Computational Geometry 14 (1–3) (1999) 119–135.
[9] A. Szymczak, D. King, J. Rossignac, An Edgebreaker-based efficient compression scheme for connectivity of regular meshes, in: Proceedings of 12th Canadian Conference on Computational Geometry, 2000, pp. 257–264.
[10] G. Taubin, J. Rossignac, Geometric compression through topological surgery, ACM Transactions on Graphics 17 (2) (1998) 84–115.
[11] C. Touma, C. Gotsman, Triangle mesh compression, in: Graphics Interface'98 Conference Proceedings, 1998, pp. 26–34.